Exploiting Efficient Densest Subgraph Discovering Methods for Big Data

Bo Wu[◇] and Haiying Shen[♯]
[◇]Electrical and Computer Engineering, Clemson University, Clemson, SC 29634
[♯]Department of Computer Science, University of Virginia, Charlottesville, VA 400740
[◇]bwu2@clemson.edu, [♯]hs6ms@eservices.virginia.edu

***Abstract*—Discovering the densest subgraph is important in graph analysis, which has wide-ranging applications from social network community mining to the discovery of biological network modules. However, the previous algorithms neglect the connectivity of the dense subgraph since it is a challenge to give consideration to both subgraph structure and time efficiency. As a result, it may lead to isolated subgraphs in the output though they aim to find one connected dense subgraph. Also, there are lack of efficient algorithms for big natural graphs, especially considering datasets become increasingly larger in this era of Big Data. Furthermore, previous algorithms fail to take advantage of various features of natural graphs (e.g., power-law degree distribution, homophyly of vertices, and power-law community size) which can be applied to improve the efficiency and precision of the densest subgraph discovery. To handle these problems, in this paper, we study the densest subgraph problem by designing two different algorithms based on different features that natural graphs have. First, by analyzing the features of natural graphs, we design a heuristic algorithm for discovering the connected densest subgraph for massive undirected graphs in a MapReduce framework by taking advantage of the features of natural graphs. Second, we propose an exact algorithm for big data for the problem of discovering the densest subgraph. Experimental results show that our algorithms are more time-efficient and precise than other algorithms.**

I. INTRODUCTION

A challenge in the analysis of complex networks is to discover densest subgraph. Densest subgraph is a subgraph of the entire graph in which the nodes have largest average degree. The densest subgraph is often interpreted as "communities" which have a lot of real applications [1**?** –6]. Various algorithms [7–10] are proposed for solving the densest subgraph problem. The most important problem in the previous algorithms [7–9] is that they ignore the connectivity of the returned densest subgraph. It means that the returned subgraph may consist of several isolated connected components that maximize its density. For example, previous algorithms [7–9] may correctly find the two isolated components as one community that are not connected to each other in Figure 1. Also, in spite of many proposed densest subgraph discovery algorithm, there are lacking of efficient algorithms for massive graphs, especially considering datasets become increasingly larger in this era of Big Data. Actually, natural graphs are completely different from random graphs. They are uniquely featured by small world phenomenon [11], power-law degree distribution [12], high clustering coefficient [13], power-law community size distribution [14], assortativity of vertices [15] and so on. Actually, these features can help us improve the precision and efficiency of the densest subgraph discovery. However, previous algorithms [7–9] rarely take advantages of these unique features of natural graphs. Therefore, it is a challenge to take advantages of the unique features in densest subgraph discovery. Another problem is that all the exact algorithms [3, 7] are in-memory algorithms which are not suitable for big data. Furthermore, the applicability of current algorithms to different kinds of graphs has not been discussed. Different kinds of natural graphs have different structure features. For example, some of the natural graphs [13] have community structures and some others do not have community structures [12]. Actually, the densest subgraph discovery is only meaningful when the natural graphs have community structure. For the graphs without community structure, the densities of different subgraphs are close to each other and the densest subgraph loses its representativeness. However, current works on exact and approximate algorithms [3, 8, 9] fail to discuss the applicability of their algorithms on different kinds of natural graphs. All the exact algorithms are in-memory algorithm which are not suitable for big data. To handle these problems, in this paper, we study the densest subgraph problem by designing two different algorithms based on different features that natural graphs have.

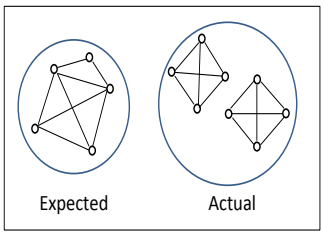


Fig. 1. An example of failing to find connected densest subgraph.

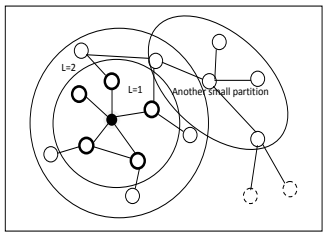


Fig. 2. An example of components when L = 1 and L = 2, respectively.

First, Guided by some useful theorems derived from the unique features of natural graphs, we design a heuristic densest subgraph discovery algorithm that is more advantageous than the previous algorithms in that it is more time and space efficient and its outcomes are more precise. In the experiments, we apply our algorithm for massive natural graphs and compare the performance with previous algorithms [8, 9] to show its superior performance.

Second, we design an exact algorithm for big data for discovering the densest connected subgraph for undirected massive graphs in a MapReduce framework. Our algorithm

has two phases. In the first phase, it carefully reduces the dataset in a MapReduce framework without loss of any nodes existing in the exact densest subgraph. In the second phase, the reduced dataset can be handled in-memory in one computer by an exact densest subgraph discovery algorithm [3]. Therefore, we can find the exact densest subgraph for big data by taking advantage of both MapReduce and in-memory computing on one computer. Furthermore, we theoretically prove the correctness of our algorithm and analyze its applicability on different complex network models [12, 16]. Finally, we conduct extensive experimental evaluations in a MapReduce framework on both massive real-world graphs and simulated graphs to test our algorithm in comparison with other algorithms. Experimental results show that our algorithm is capable of discovering the connected densest subgraph for big data. This algorithm can be used to find the exact densest subgraph with losing some time efficiencies.

The rest of this paper is organized as follows. Section II presents the related work. Section III describes the design of our heuristic densest subgraph discovery algorithm, the analysis of the algorithm and extensive experimental evaluation on massive real-world graphs and simulated graphs. Section IV describes the design of our exact densest subgraph discovery algorithm for big data, the analysis of the algorithm and extensive experimental evaluation on massive real-world graphs and simulated graphs. Section VI summarizes the paper with remarks on our future work.

II. Related Work

Dense subgraph: Goldberg [3] formally introduced the original problem of discovering the densest subgraph in an undirected graph and gave an algorithm that requires $O(\log n)$ running time (n is the number of vertexes in the graph) to find the optimal solution based on the min-cut technique. Khuller *et al.* [17] proposed a similar polynomial time complex algorithm for discovering densest subgraph in a directed graph. Several subproblems were derived from the traditional densest subgraph problem. Feige *et al.* [18] defined and studied the dense k-subgraph maximization problem, which is applied to find a set of k vertices with maximum average degree in the subgraph. Asahiro *et al.* [19] defined and studied the problem of discovering a k-vertex subgraph of a given graph G that has at least $f(k)$ edges. Saha *et al.* [7] defined the densest subgraph problems with distance (i.e., the diameter of output subgraph) restrictions and specific subset restrictions (a specific subset must be in the output), and provided algorithms for them. Also, they studied the problem of discovering the multiple almost densest subgraphs [7]. When it comes to big data era, due to the complexity of dense subgraph problem, various approximate and heuristic algorithms were designed to meet the computing time and space challenges. Charikar [8] described a simple greedy algorithm and showed that it leads to a 2-approximation to the optimum. This algorithm was improved by Bahmani *et al.* [9], which leads to within a factor $2(1+\varepsilon)$ of the optimum in a MapReduce framework. Samir *et al.* [17] developed fast polynomial time algorithms for several variations of dense subgraph problems for both directed and undirected graphs. Some heuristic algorithms [20, 21] were also designed based on different techniques. Gibson *et al.* [20] solved the discovering dense subgraphs problem based on a shingling technique, while Chen *et al.* [21] solved the same problem by the matrix blocking technique.

Features of natural graphs: Psychologist Stanley Milgram conducted a series of experiments that indicated the average path length of peoples in human society is short [11], which is called *small world phenomenon*. Besides the small world phenomenon, evidences suggest that in most real-world networks, and particular social networks, nodes tend to create tightly knit groups characterized by a relatively high density of ties; this likelihood tends to be greater than the average probability of a tie randomly established between two nodes [13]. A *community* is a cohesive subset of nodes with denser inner links, relatively to the rest of the network [1]. Previous studies have shown that the degree distribution of many real-world networks follow a power-law [12]. Also, not only the degrees follow a power-law, but also there is a scale-free distribution of communities [14].

However, the previous dense subgraph discovery algorithms neglect the connectivity, which may lead to isolated sever components in the output subgraph, which are deviated from the original goal of connected subgraph discovery. Also, the previous algorithms are not efficient enough and scalable to handle massive datasets, which is very common in this Big Data era. Also, they fail to leverage the unique features of natural graphs, which otherwise can be used to improve the efficiency and precision of the algorithms. Another problem is that all the exact algorithms are in-memory algorithms which are not suitable for big data. Unlike these previous algorithms, our work can handle these problems. We study the densest subgraph problem by designing two different algorithms based on different features that natural graphs have. First, by analyzing the features of natural graphs, we design a heuristic algorithm for discovering the connected densest subgraph for massive undirected graphs in a MapReduce framework by taking advantage of the features of natural graphs. Second, we propose an exact algorithm for big data for the problem of discovering the densest subgraph.

III. Heuristic Dense Subgraph Discovery Algorithm

In this section, we design a heuristic densest subgraph discovery algorithm for undirected natural graphs. Our aim is not only to design the algorithm itself, but also to verify the unique features of natural graph structures. To be more specific, if our heuristic algorithm performs better, it can be used as an evidence to verify our theorems, and further the unique features of natural graph structure.

A. Theoretical Analysis

We list the previously observed features of natural graph structures which can benefit our algorithm design later as follows: i) The degree of vertices follows a power-law distribution in natural graphs [12]; ii) The community size follows a power-law distribution in natural graphs [14]; iii) The vertices

tend to build connection with other vertices which have similar degree [15], which we call assortativity.

Then, based on these features, we propose two theorems, which lay the foundation of our proposed dense subgraph discovery algorithm.

Suppose the vertices in the same dense subgraph strictly have the same degree, and the size of the dense subgraphs follows a power-law distribution [16], then we have Theorem 3.1 as follows.

Theorem 3.1: Suppose a graph $G = (V, E)$ with a degree power-law $X_d \propto d^{-\gamma}$, where X_d is the number of vertices of degree d and γ is the power-law exponent, then we have the maximum dense subgraph size $s = n^{\frac{1}{\gamma+1}}$, where $n = |V|$.

Proof 1: Suppose there is a dense subgraph with size s^*, then we have $X_{s_*} = n{s_*}^{-\gamma}$. Since the vertices in same dense subgraph strictly have the same degree, then it requires $n{s_*}^{-\gamma} \geq s_*$. Therefore, we have $n{s_*}^{-(\gamma+1)} \geq 1$. Finally, we have the maximum dense subgraph size $s = n^{\frac{1}{\gamma+1}}$.

Based on Theorem 3.1, we can estimate that for a graph with a million vertices and a power-law exponent $\gamma = 2$, the dense subgraph with maximum size is with about 100 vertices, which is quite small. Based on Theorem 3.1, we derive Lemma 3.1 as follows.

Lemma 3.1: For a maximum dense subgraph with size $s = n^{\frac{1}{\gamma+1}}$, the diameter of the dense subgraph equals 1.

Proof 2: When size $s = n^{\frac{1}{\gamma+1}}$, we have $X_s = ns^{\gamma} = s$. Therefore, each pair of the vertices are connected. Therefore, we have the diameter of the dense subgraph equals 1.

All the above analysis is based on the assortativity feature of natural graphs. Not all the natural graphs are assortative and there are some disassortative natural graphs in which the vertices tend to connect with other vertices have different degree. However, when it comes to the community, all of the natural graphs which have community structure are assortative natural graphs [22, 21, 20, 7]. Disassortative natural graphs usually do not have the community feature since the community itself is based on the effect of assortativity. Therefore, disassortative natural graphs are not in our consideration. To be more specific, in this paper, the natural graphs indicate disassortative natural graphs.

B. A High-Level Description

Intuitively, we assume that we can get all the vertices from one vertex in the dense subgraph by traversing on the edges in a small number of steps since dense subgraphs have very small diameters. The start points of the traversing should be the centers of the dense subgraphs. Although the subgraphs retrieved by traversing may contain some vertices that do not belong to the dense subgraphs, we can still easily eliminate it by the traditional method in [3] if the size of the subgraph is small enough. In other words, this method partitions the intractable big dataset to many small components, which still contain the dense subgraphs, but can be processed easily by traditional method [3], which assume that the memory is large enough to hold all the data.

Based on this rationale, we present a densest subgraph discovery algorithm. The algorithm partitions the graphs into overlapping small components, which contain the dense subgraphs. An example is shown in Figure 2. We then use traditional densest subgraph discovery algorithm [3] to discover the dense subgraphs in the small components very quickly since each entire small component can be stored in memory. Next, we measure the densities of the dense subgraphs which have been found in each small component to discover the densest subgraph and top dense subgraphs. We prove that this algorithm achieves high efficiency for the densest subgraph and dense subgraphs problems in the experiment section. Besides, our algorithm can guarantee the connectivity of the output subgraph since it considers structure of the graphs.

C. Parameter Determination

We need to determine three parameters for our algorithm: i) the number of traversing steps, ii) the criteria of seed selection (The seeds are the vertices chosen for further partition), and iii) the number of seeds. Below, we explain the details of the parameter determination.

The number of traversing steps (denoted by L): The quality of a subgraph is measured by the density of the subgraph; higher density means higher quality. If L is too large, then the size of each component will be too large; if L is too small, then we may not discover high-quality dense subgraphs. However, Theorem 3.1 shows that the densest subgraph of a natural graph should have a very small diameter. Therefore, we infer that setting L to 1 is enough to discover qualified densest subgraphs since we can traverse all the vertices from any vertex within at most 2 steps in these subgraphs. Figure 2 shows an example of the components when $L = 1$ and $L = 2$, respectively.

Criteria of seed selection: Suppose we can properly set the value L, then the selected seeds are the main factor for the effectiveness of the algorithm in discovering dense subgraphs. An ideal method is to select the most highly connected vertex from each dense subgraph so that we can discover each dense subgraph non-repeatedly. However, this task is non-trivial since we do not know the places of the dense subgraphs in the graph, or whether two vertices are in the same dense subgraph. Therefore, we use a heuristic method which chooses vertices with higher degrees as seeds since high-degree vertices are more likely to be in dense subgraphs than low-degree vertices. However, for a large graph, choosing a vertex with the highest degree may lead to a memory overload, considering the small world feature of natural graphs. Also, based on Theorem 3.1, we know that the densest subgraph should be small and we only need components with a size of order of 100 even for a graph with a million vertices as the previous example in Section IV-C. Therefore, we select the seeds which have the suitable degrees as Theorem 3.1 indicated so that it will be with high probability to find the densest subgraph. For the degree power law exponent γ for each natural graph, it can be easily estimated by a Kolmogorov-Smirnov (K-S) test which is used to determine if two datasets differ significantly.

The number of seeds (denoted by m): If the theoretical analysis is absolutely true, then we only need to choose one vertex to get the densest subgraph since we can exactly find a

vertex in the densest subgraph. However, we should recognize that there must be a randomness in the real world. Therefore, we may need to choose more than one vertex, and compute the densest subgraph from multiple component candidates. Fortunately, for a graph in which the degree follows a power-law, Theorem 3.1 guarantees that even if the graph is large, there are a limited number of qualified seeds since the densest subgraph is small and each vertex in the densest subgraph has a narrow range of degree. For example, for a graph with a million vertices and has a degree power-law exponent $\gamma = 2$, there are only about 100 vertices with degree about 100. Also, for the MapReduce framework, a limited number of parallel processes do not influence the time efficiency. Therefore, we can select multiple seeds without at the cost of degraded algorithm efficiency by taking advantage of the MapReduce framework.

Since our algorithm is heuristic, in Section V-C, we experimentally study the influence of the parameters on the effectiveness of the densest subgraph discovery algorithm, and find that a floor level of parameters, i.e., setting L to 1, and m to 0.01% of all the vertices, can guarantee a good performance.

D. Process of the Algorithm

Our dense subgraph discovery algorithm has two phases: 1) graph partition, and 2) densest subgraph discovery. Step 1 has the following two sub-phases: i) Sub-phase 1: we measure the degree of each vertex, and select the top m suitable degree vertices as seeds; ii) Sub-phase 2: we partition the large graph to the small components, which are generated by traversing the graph from the seeds for L steps. Then we can discover the densest subgraph in the small components. In Step 2, we can apply any traditional algorithms to discover the densest subgraph contained in the small components.

Our algorithm can be easily implemented in parallel computing frameworks such as MapReduce. Below, we introduce the details of phase 1: graph partition.

Algorithm 1: Seed selection in a MapReduce program

```
Mapper
    Input: < u, v >
    emit < u, v >;
end
Reducer
    Input: < u, neighbor_list >
    if |neighbor_list| > threshold then
        emit < u, $ >;
    end
end
```

Figure 3 illustrates the flowchart of the graph partition. Algorithm 1 shows the process of seed selection in the first step in Figure 3. The input of this MapReduce program is the edge list, which records each edge (u, v) in the graph twice by (u, v) and (v, u) (u and v are the two endpoints of the edge). If vertex u is selected as a seed, then we add an output $< u, \$ >$, which will be used in the later part of the algorithm. The parameter m can be determined by different realistic demands.

Algorithm 2 shows the process of tagging seeds in each edge in the initial edge list, and it outputs the dataset with seeds tagged with *. Then, in Algorithm 3, we use two MapReduce rounds to traverse one more hop from the seeds, and tag all the edges whose two endpoints both belong to the vertex set in the traversal. In the first MapReduce round, we tag the edges in the traversal path. In the second MapReduce round, we tag the remaining edges whose both two endpoints belong to the vertex set in the traversal.

E. Densest Subgraph Discovery

After we partition the large graph to small components, discovering the densest subgraph has become easy since each small component can be stored in memory and handled easily by traditional algorithms. In this paper, we can simply use *ExactAlg* [3] which is a classic polynomial algorithm to find the exact densest subgraph. *ExactAlg* is an exact polynomial time algorithm for discovering the densest subgraph for an undirected graph based on a min-cut max-flow technique. The algorithm transfers the densest subgraph problem to a series of min-cut problems and finds the densest subgraph by recursively constructs a new graph based on the initial graph and find the min-cut (S, T) on the new graph. Finally, the densest subgraph can be derived from the min-cut (S, T). The detailed design is presented in [3].

Algorithm 2: Tag seeds in each edge

```
Mapper
    Input: < u, v > and < u, $ >
    emit Input;
end
Reducer
    Input: < u, neighbor_list >
    if $ ∈ |neighbor_list| then
        neighbor_list = neighbor_list\$;
        foreach v in neighbor_list do
            emit < u*, v >;
        end
    else
        foreach v in neighbor_list do
            emit < u, v >;
        end
    end
end
```

Algorithm 3: Traverse 1 more hop based on current component

```
1: Mapper
    Tag the edges in the traversal path.
   end
   Reducer
    Generate the edge list.
   end
2: Mapper
    Tag the remaining edges whose two endpoints both belong the
    vertex set in the traversal.
   end
   Reducer
    Generate the edge list.
   end
```

Since the small components are data independent from each other, we can process them in parallel in MapReduce. To sum up, the entire algorithm for the densest subgraph discovery in each small component is shown in Algorithm 4. The key of the data in the MapReduce framework is the small component

ID, and the value of the data in the MapReduce framework is the edge list of the corresponding small component. In the Reduce function of MapReduce, we discover the candidates of the densest subgraph in each small component by *ExactAlg* in order to guarantee connectivity and improve the accuracy. Finally, we select the densest subgraph from the candidates.

Algorithm 4: Densest subgraph discovery

```
1: Mapper
   | Input: < seedID, edge_list >
   | emit Input;
   end
   Reducer
   | Input: < seedID, edge_list >
   | use ExactAlg to discover the densest subgraph in edge_list;
   end
```

IV. Exact Densest Subgraph Discovery Algorithm

In this section, we design an exact algorithm for discovering the densest connected subgraph for big data for undirected massive graphs in a MapReduce framework. Our algorithm has two phases. In the first phase, it carefully reduces the dataset in a MapReduce framework without loss of any nodes existing in the exact densest subgraph. In the second phase, the reduced dataset can be handled in-memory in one computer by an exact densest subgraph discovery algorithm [3]. Therefore, we can find the exact densest subgraph for big data by taking advantage of both MapReduce and in-memory computing on one computer. As a result, the first and second problems mentioned above can be resolved. Furthermore, we theoretically prove the correctness of our algorithm and analyze its applicability on different complex network models [12, 16]. Finally, we conduct extensive experimental evaluations in a MapReduce framework on both massive real-world graphs and simulated graphs to test our algorithm in comparison with other algorithms. Experimental results show that our algorithm is capable of discovering the connected densest subgraph for big data.

A. The Design of the Algorithm

The natural graphs usually follow a power-law degree distribution [12]. Intuitively, for graphs with such a feature, most of the vertices with low degrees have very small probabilities to be in the densest subgraph. Therefore, the basic idea of the M-O algorithm is trying to reduce the initial size of the dataset by deleting the vertices with very low degrees in order to fit the reduced dataset in the memory of one computer. Then, the algorithm applies the min-cut max-flow technique [3] to find the densest subgraph in the remaining graph on one computer. With only one round of the MapReduce process and in-memory computing on one computer without the data transfer between computers, the M-O algorithm achieves high time-efficiency.

Based on this intuition, the M-O algorithm has two phases: 1) the graph reduction phase, and 2) densest subgraph discovery phase. Figure 3 illustrates the flowchart of the M-O algorithm. Algorithm 5 presents the pseudocode of the

Algorithm 5: The pseudocode of the M-O algorithm.

```
1: Given: G = (V, E);
2: S ← V, ρ_max ← ρ(S);
3: while S > threshold do
   | S_c ← {v_i ∈ S | deg_S(v_i) ≤ ρ_max};
   | S ← S \ S_c;
   | if ρ(S) > ρ_max then
   |   | ρ_max ← ρ(S);
   | end
   end
4: G_0 = (S_0, E(S_0)) ← G_S = (S, E(S));
5: Given: G = (V, E);
6: l ← 0, u ← n;
7: while (l − u) ≥ 1/(n(n−1)) do
   | g ← (l+u)/2;
   | Construct N = (V_N, E(V_N));
   | Find min-cut (S, T);
   | if S = {s} then
   |   | u ← g
   | end
   | if S ≠ {s} then
   |   | l ← g;
   |   | V_1 ← S − {s};
   | end
   end
8: return subgraph of G derived by c(S, T);
```

M-O algorithm. For a given $G = (V, E)$, we first delete all the vertices, which have degrees equal or smaller than the maximum density of remaining graph during the deleting process streamingly (blocks 1-3). After the size of the datasets is reduced to a suitable size (i.e., threshold in Figure 3) that can fit into the memory of a computer for the densest subgraph discovery phase, we apply min-cut max-flow technique to find the densest subgraph in the remaining graph (blocks 5-7). Block 8 returns the results. In Section IV-C, we prove the correctness and applicability of the M-O algorithm.

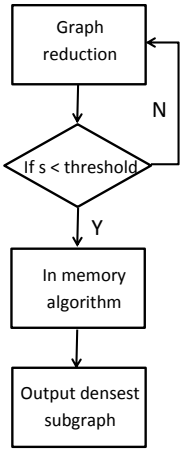


Fig. 3. Flowchart of the M-O algorithm

Next, we introduce the min-cut max-flow based exact densest subgraph discovering algorithm. The edges in the initial graph are assigned with capacity 1. A min-cut of a graph is a cut (a partition of the vertices of a graph into two disjoint subsets) whose cut edge set (consisting of edges that cross the two disjoint subsets) has the smallest sum of capacities. We use $c(S, T)$ to denote the min-cut of a graph that splits the graph to two partitions, S and T. The capacity of min-cut $c(S, T)$ is the sum of the capacities of the cut edge set. Then, the densest subgraph can be found by recursively constructing a new graph based on the current graph and finding the min-cut $c(S, T)$ on the new graph. In each iteration of the recursion, we first construct a new graph by adding two vertices, s and t. We build an edge between each vertex v_i and s and assign each edge with capacity e, where e is the number of edges in the initial graph. We also build an edge between each vertex v_i

and t and assign each edge with capacity $e+2g-d_{v_i}$ where g is an estimated value of the density of the densest subgraph and d_{v_i} is the degree of v_i in the initial graph. Second, we update the upper bound p of g by the current value of g if $S = \{s\}$ and update the lower bound l of g by the current value of g if $T = \{t\}$. Third, we assign g with value $\frac{p+l}{2}$ and start the next iteration. The recursion is stopped when $(l-p) < \frac{1}{n(n-1)}$ where n is the number of vertices in the new graph. Then, the partition $S - \{s\}$ is the densest subgraph. For more details of the min-cut max-flow technique, please refer to [3].

B. Implementation on MapReduce

In the graph reduction phase, we delete a batch of vertices every time, which can be implemented in parallel computing frameworks such as MapReduce [23]. MapReduce is running based on the data structure in $< key; value >$ pairs. Current densest subgraph algorithm [9] takes two rounds of MapReduce processes for deleting the vertices with degrees smaller than a certain value. In the first round, it tags all the vertices that need to be deleted. Then, in the second round, it deletes all the tagged vertices. Our algorithm is more time-efficient in that it can delete the vertices with degrees smaller than a certain value in one round of MapReduce process as shown in Algorithm 6.

Algorithm 6: MapReduce implementation of the graph reduction

```
1: Mapper
   Input: < v; u > and (or) < v; ũ >
   emit Input;
   end
   Reducer
   Input: < v; neighborlist >
   clean the neighbor list;
   calculate the degree of v;
   if degree(v) > threshold then
      foreach v in neighborlist do
         emit < v; u_i >;
      end
   else
      foreach v in neighborlist do
         emit < u_i; ṽ >;
      end
   end
   end
```

The input of the first round of our MapReduce program is the edge list, which records each edge connecting vertices u and v in the graph twice; that is, $< u; v >$ and $< v; u >$. The Map function outputs the neighbor list of each vertex in the form of $< v; u_1, u_2, ..., u_n >$, where v is an arbitrary vertex and $u_1, u_2, ..., u_n$ are the other ends in the neighbor list of vertex v (block 1). The Reduce function outputs edges in the form of $< v; u_1 >, < v; u_2 >, ..., < v; u_n >$ if the degree of v is larger than the threshold, which is ρ_{max} in Algorithm 5. Otherwise, it outputs edges in the form of $< u_1; \widetilde{v} >, < u_2; \widetilde{v} >, ..., < u_n; \widetilde{v} >$. $\widetilde{v}$ indicates that vertex v should be deleted later on in the neighbor list of each node u_i $(i = 1, 2, ..., n)$ (block 2).

This output of the Reduce function is the input of the next round of our MapReduce program, which consists of the edge list and the edges that should be deleted recorded as $< u; \widetilde{v} >$. In the next round, the Map function outputs the neighbor list of each vertex in the form of $< v; u_1, u_2, ..., u_n, \widetilde{u}_i >$, where $\widetilde{u}_i$ indicates that vertex u_i should be deleted (block 1). In the Reduce function, for an input of neighbor list of a vertex v, we first delete the vertices which have been tagged as $\widetilde{u}$ in last round in the neighbor list of vertex v and reproduce its neighbor list in the remaining graph. Then, we output edges in the form of $< v, u_1 >, < v, u_2 >, ..., < v, u_n >$ if the degree of v is larger than the threshold. Otherwise, we output edges in the form of $< u_1; \widetilde{v} >, < u_2; \widetilde{v} >, ..., < u_n; \widetilde{v} >$ (block 2). This process of the round repeats until the size of the data is suitable for in-memory computing.

Our MapReduce strategy only uses one MapReduce round for deleting the vertices tagged in the previous round and tagging the vertices that should be deleted in the next round at the same time. Therefore, our algorithm only uses one MapReduce round for one recursion of data reduction.

The M-O algorithm applies advanced MapReduce strategy for the graph reduction phase, so that it can further improve the time efficiency.

C. Theoretical Analysis

In this section, we analyze the correctness and applicability of the M-O algorithm. There are a number of requirements that the M-O algorithm needs to meet to verify its correctness and applicability. The requirement 1 of the correctness is that the process of the graph reduction cannot delete any vertices in the densest subgraph; the requirement 2 of the correctness is that the final densest subgraph discovered by the M-O algorithm must be connected. The requirement 1 of the applicability is that the graph size must be reduced to a suitable size that can fit in the memory of one computer for in-memory computing; the requirement 2 of the applicability is that the number of rounds of the graph reduction should be as small as possible to achieve high time-efficient. Furthermore, the performance of the M-O algorithm is dependent on the graph topology. Therefore, we further discuss the applicability of the M-O algorithm to different kinds of natural graphs.

1) Correctness of the reduction: First, we prove that the strategy of the graph reduction of the M-O algorithm does not delete any vertices in the densest subgraph.

Lemma 4.1: Suppose $G_S = (V_S, E(V_S))$ is the densest subgraph of graph G, then we have $deg_{V_S}(v_i) \geq \rho(V_S)$ for any vertex $v_i \in V_S$.

Proof 3: We prove it by contradiction. Suppose there is at least one vertex $v_i \in V_S$ and $deg_{V_S}(v_i) < \rho(V_S)$, then we delete vertex v_i from G_S, and get graph $G_{S^-} = (V_S \backslash \{v_i\}, E(V_S \backslash v_i))$. The density of graph G_{S^-} is:

$$\begin{aligned} \rho(V_S \backslash \{v_i\}) &= \frac{|E(V_S \backslash v_i)|}{|V_S \backslash \{v_i\}|} \\ &= \frac{\rho(V_S)|V_S| - deg_{V_S}(v_i)}{|V_S| - 1} \end{aligned}$$

Since $deg_{V_S}(v_i) < \rho(V_S)$, we have:

$$\rho(V_S \backslash \{v_i\}) > \frac{\rho(V_S)(|V_S| - 1)}{|V_S| - 1} > \rho(V_S)$$

This contradicts with the precondition. Therefore, we have $deg_{V_S}(v_i) \geq \rho(V_S)$ for any vertex $v_i \in V_S$.

Theorem 4.1: After we delete all the vertices with degrees no more than the maximum density of the remaining graph of G in block 3 of Algorithm 5 to obtain G_S, the densest subgraph of G is in G_S.

Proof 4: We prove it by contradiction. Suppose there is one densest subgraph $G_x = (V_x, E(V_x))$, where $G_x \not\subset G_S$, then there is a vertex subset I where $I \subset V_x$ and $I \not\subset V_s$. There is a time in the deleting process that the first vertex v_i in I is deleted from the current V_s (denoted by ${V_s}^+$). Therefore, we have:

$$\rho(V_x) \leq deg_{V_x}(v_i) \leq deg_{{V_s}^+}(v_i) < \rho({V_s}^+)$$

This implies that $\rho(V_x) < \rho({V_s}^+)$, and at this moment, we have $V_x \subset {V_s}^+$. Hence, from the definition of the densest subgraph, we know that G_x is not a densest subgraph of G. This contradicts with the assumption. Therefore, for any densest subgraph G_x of G, we have $V_x \subset V_S$.

Theorem 4.1 guarantees that the graph reduction in the M-O algorithm cannot delete any vertices in the densest subgraph.

2) Connectivity of the solution:

Theorem 4.2: Suppose $G_1 = (V_1, E(V_1))$ is the densest subgraph of graph G discovered by the M-O algorithm where G could be a connected or disconnected graph, then we can get that $G_1 = (V_1, E(V_1))$ is a connected graph.

Proof 5: We prove it by contradiction. Suppose $G_1 = (V_1, E(V_1))$ is a disconnected graph which consists of two isolated subgraphs, $G_3 = (V_3, E(V_3))$ and $G_4 = (V_4, E(V_4))$, then we have the capacity of the min-cut from which we derived G_1 equals:,

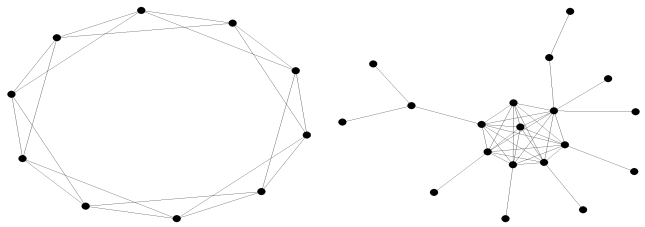
$$\begin{aligned} c(S,T) &= \sum_{i\in S, j\in T} c_{ij} \\ &= m|V_2| + (m|V_1| + 2g|V_1| - \sum_{i\in V_1} d_i) + \sum_{i\in V_1, j\in V_2} c_{ij} \\ &= m|V| + 2|V_3|(g - \frac{\sum_{i\in V_3} d_i - \sum_{i\in V_3, j\in V_2} 1}{2|V_3|}) \\ &+2|V_3|(g - \frac{\sum_{i\in V_4} d_i - \sum_{i\in V_4, j\in V_2} 1}{2|V_4|}) \end{aligned}$$

where V_2 is the other part of the min-cut. Since g is the density of $\rho(V_1)$, suppose $\rho(V_3) \leq \rho(V_4)$, then we have:

$$\begin{aligned} c(S, T \cup V_3) &= m|V| + 2|V_3|(g - \frac{\sum_{i\in V_3} d_i - \sum_{i\in V_3, j\in V_2} 1}{2|V_3|}) \\ &< c(S,T) \end{aligned}$$

Since the min-cut from which we derived G_1 should be a min-cut in the densest subgraph discovery phase of the M-O algorithm, this contradicts with the precondition. Therefore, $G_1 = (V_1, E(V_1))$ is a connected graph.

Theorem 4.2 guarantees that the densest subgraph discovered by the M-O algorithm is connected.



(a) Unsuitable for reduction (b) Suitable for reduction

Fig. 4. Examples of different graph topologies

3) Applicability of the M-O algorithm: There are two main factors that influence the time efficiency of the M-O algorithm. One is the number of rounds required for the graph reduction. The other is the remaining graph size after the graph reduction. These two factors are closely related to the topology of the initial graph. For example, the graph in Figure 4(a) cannot be reduced, while the graph in Figure 4(b) only takes one round for a great size reduction. In this section, we analyze the relationship between the performance of the M-O algorithm and the topologies of natural graphs. In order to study the relationship, we use the BA network [12] and BTER network [16] (which are two typical complex networks) for our study. The BA network has the feature of power law degree distribution but has no community structures. The BTER network not only has the feature of power law degree distribution, but also has the community structures. We theoretically analyze the performance of the M-O algorithm on the BA network and BTER network to show its performance on the complex networks with or without community structures.

Theorem 4.3: Suppose $G = (V, E(V))$ is a BA network and $G_S = (V_S, E(V_S))$ is the densest subgraph of BA network G, then we have

$$\rho(V_S) = \rho(V) = m.$$

where m is the number of edges each vertex created when the vertex first joined the network.

Proof 6: Suppose there is a process that we can get the densest subgraph $G_S = (V_S, E(V_S))$ by deleting vertex set $V_d = \{v_1, v_2, v_3, v_4, ..., v_n\}$, then we have:

$$\begin{aligned} \rho(V_S) &= \frac{|E(V_S)|}{|V_S|} \\ &= \frac{|E(V)| - \sum_{v_i\in V_d} \deg_d(v_i)}{|V| - |V_d|} \end{aligned}$$

where $\deg_d(v_i)$ is the degree of v_i when v_i is deleted from the remaining graph.

Since each vertex created m edges with other vertices when the vertex first joins the network, we have $\sum_{v_i\in V_d} \deg_d(v_i) \geq m|V_d|$. Therefore, we have:

$$\rho(V_S) = \frac{|E(V_S)|}{|V_S|} \leq m$$

Also, when the network tends to be very large, we have:

$$\rho(V) = \frac{|E(V)|}{|V|} = \frac{m|V|}{|V|} = m$$

Therefore, we have:

$$\rho(V) = \rho(V_S) = m$$

Since the density of the densest subgraph is equal to the density of the whole network, there are no denser subgraphs in the network. Therefore, Theorem 4.3 indicates that BA network has no communities based on the density criteria. Then, it is meaningless to discover the densest subgraph in a BA network. Also, from the definition of BA network, we know that each vertex has a degree at least m since each vertex created m edges once it participates in the network. Therefore, the graph reduction in the M-O algorithm is useless for reducing the size of a BA network since all the vertices have a degree equal or larger than the density of the densest subgraph. Therefore, in Section V-C, we only evaluate our algorithms on the graphs which contain densest subgraph. For BA networks, we can directly return the whole graph as the densest subgraph without any computation based on Theorem 4.3.

Lemma 4.2: Suppose $G = (V, E(V))$ is a BTER network which approximately satisfies $\log y = \beta - \gamma \log x$, where x is the degree of the vertices with the same degree and y is the number of vertices with degree x, then we can delete $\frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{\zeta(\gamma)}} \frac{1}{x^\gamma}}{\zeta(\gamma)}$ of the total vertices in the first round of the graph reduction. Here, γ is the exponent parameter of degree distribution of the BTER network and $\zeta(\gamma)$ is the Riemann zeta function [24] of γ.

Proof 7: Since $0 \leq \log y = \beta - \gamma \log x$, we have $x \leq e^{\frac{\beta}{\gamma}}$. Then, we have:

$$\begin{aligned}\rho(V) &= \frac{|E(V)|}{|V|} \\ &= \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma} \times x}{2\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma}} = \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{1}{x^{\gamma-1}}}{2\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{1}{x^\gamma}}\end{aligned}$$

When $e^{\frac{\beta}{\gamma}} \to +\infty$, according to the Riemann zeta function [24], we have:

$$\rho(V) = \frac{\zeta(\gamma-1)}{2\zeta(\gamma)}$$

The number of the vertices with degrees smaller than $\rho(V)$ equals $\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma} \times x$. Therefore, the percent of deleted vertices in V equals:

$$\frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}} \frac{e^\beta}{x^\gamma}}{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma}} = \frac{\sum_{x=1}^{\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}} \frac{1}{x^\gamma}}{\zeta(\gamma)}$$

Theorem 4.4: Suppose $G = (V, E(V))$ is a BTER network which approximately satisfies $\log y = \beta - \gamma \log x$, where x is the degree of the vertices and y is the number of vertices, then we can delete

$$\frac{\sum_{x=1}^{(\zeta(\gamma-1)-\sum_{x=1}^{\rho_{-1}(V)} x^{1-\gamma})/(\zeta(\gamma)-\sum_{x=1}^{\rho_{-1}(V)} x^{-\gamma})} x^{-\gamma}}{\zeta(\gamma) - \sum_{x=1}^{\rho_{-1}(V)} x^{-\gamma}}$$

of the total vertices in the ith round of the graph reduction where $\rho_{-1}(V)$ is the $\rho(V)$ in the $(i-1)$th round.

Proof 8: Since vertices with a same degree form into an isolated subgraph in a BTER network, then we can consider that the deleted vertices do not influence the degrees of the remaining vertices. Therefore, we have:

$$\begin{aligned}\rho(V) &= \frac{\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma} \times x - \sum_{x=1}^{\varepsilon} \frac{e^\beta}{x^\gamma} \times x}{2(\sum_{x=1}^{e^{\frac{\beta}{\gamma}}} \frac{e^\beta}{x^\gamma} - \sum_{x=1}^{\varepsilon} \frac{e^\beta}{x^\gamma})} \\ &= \frac{\zeta(\gamma-1) - \sum_{x=1}^{\varepsilon} x^{1-\gamma}}{2(\zeta(\gamma) - \sum_{x=1}^{\varepsilon} x^{-\gamma})}\end{aligned}$$

where $\rho(V)$ is the current density.

Therefore, the percent of deleted vertices in V equals:

$$\frac{\sum_{x=1}^{(\zeta(\gamma-1)-\sum_{x=1}^{\varepsilon} x^{1-\gamma})/(\zeta(\gamma)-\sum_{x=1}^{\varepsilon} x^{-\gamma})} x^{-\gamma}}{\zeta(\gamma) - \sum_{x=1}^{\varepsilon} x^{-\gamma}}$$

In the real-world graphs, γ is usually around 2 [1]. Therefore, $\frac{\zeta(\gamma-1)}{2\zeta(\gamma)}$ is very large since $\zeta(1) = +\infty$ and $\zeta(2) \approx 1.645$. Theorem 4.4 indicates that a great percentage of vertices can be deleted in the first few rounds of the graph reduction and also the graph reduction is decreasingly efficient as the percentage of deleted vertices increases. For example, when $\gamma = 1.2$, about $\frac{1+\frac{1}{2^{2.2}}}{1.49} \approx 0.82$ of the vertices are deleted in the first round of the graph reduction. This result is also consistent with the experiment result in Section V-C.

From our analysis, we have the following conclusions: i)The complex network without community structures (e.g., BA networks) is uniformly distributed, which makes the densest subgraph discovery meaningless since the density of the densest subgraph equals the density of the whole network in such kind of complex networks; ii) For the complex network with community structures (e.g., BTER networks), the M-O algorithm not only can reduce such kind of complex networks to a suitable size that can be fitted in memory, but also can reduce it to a suitable size in only a few rounds.

V. Performance Evaluation

A. Experiment Configurations

1) Environment: We run the M-O algorithm on Hadoop [25] with 4 PCs; each PC is equipped with 2.1GHz Intel core i3 processor with 2 cores, and a 2GB memory. The M-O algorithm was implemented in Python. We used the datasets in Table I in the experiments, which are from the SNAP library [26].

2) Datasets: We classify the datasets to two different types (small and large); small datasets can be fitted in the memory of one computer and large datasets are too large to be fitted in memory. We would like to see the different performances of the M-O algorithm on small datasets and large datasets. The table lists the name, the description, the number of vertices ($|V|$), the number of edges ($|E|$) and the type of each dataset. In the datasets, *Wiki-Vote* is the only directed graph. However, directed graphs have similar natural graph features as undirected graphs [27]. Therefore, we just treat *Wiki-Vote* as an undirected graph in order to enrich our datasets as the current work [28].

3) Algorithms for comparison: The other algorithms used for comparing with our algorithms in the following evaluation part are *Approx* (greedy algorithm) and *ApproxMR* (parallel greedy algorithm). *Approx* [8] is a sequential greedy algorithm for discovering the densest subgraph for an undirected graph G. It greedily deletes the vertex with the smallest degree one by one and records the density of the remaining graph until the remaining graph is empty. The densest remaining graph in this deleting process is the discovered densest subgraph. It has been proved that *ApproxMR* can guarantee a 2-approximation for $\rho^*(S)$. *ApproxMR* [9] is an extension of *Approx* which is implemented for parallelism in MapReduce. Instead of deleting the vertex one by one, *ApproxMR* deletes a batch of vertices with degrees smaller than $(2+\varepsilon)$ times of the current density of the remaining graph in every two rounds of MapReduce process and records the current density of the remaining graph. After all the vertices are deleted from the initial graph, we obtain the densest subgraph from the records. It has been proved that *ApproxMR* can guarantee a $(2+\varepsilon)$-approximation for $\rho^*(S)$.

B. Performance of Heuristic Dense Subgraph Discovery Algorithm

The heuristic dense subgraph discovery algorithm is implemented in Python. The numbers of Mappers and Reducers in MapReduce are set manually. We use the datasets in Table I in the experiments. Since we tag the vertices with small component ID, there is an extra external memory for storing them in hard disk. Therefore, we need to evaluate the extra external memory for our algorithm. We evaluate our algorithm with focuses on two aspects. We first evaluate the extra external memory, running time, the densities of the discovered subgraphs of our algorithm with different parameters. We then compare these performance metrics of our algorithm with *Approx* and *ApproxMR*.

1) Effect of parameters on performance: External memory is not a bottleneck nowadays for computing environment. However, the linearly producing extra data produced increases the I/O costs, which is a core consideration for a MapReduce algorithm. The efficiency of an algorithm is determined by the running time, extra external memory usage, the number of discovered qualified dense subgraphs. In this section, we study the following problems: i) How does the number of traversing steps (L) influences the efficiency of the algorithm? ii) How does the number of the seeds (m) influences the efficiency of the algorithm?

Recall we select top suitable degree vertices as seeds. This process can guarantee that the components found by seeds have suitable sizes. We first used the top 0.01% of suitable degree vertices as the seed and measured the extra external memory with the increasing of parameter L. We define *extra memory ratio* as the ratio of the extra external memory divided by the memory used to store the initial datasets. Figure 5 shows the extra memory ratio versus L. As the figure shows, the extra external memory increases sharply with the increasing of L in the beginning due to the small world phenomenon [11], in which any two vertices can reach each other in a limited steps along the edges. The increasing speed of individual extra

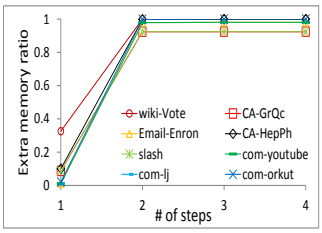


Fig. 5. Extra memory ratio vs. # of transverse steps

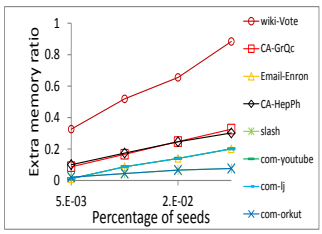


Fig. 6. Extra memory ratio vs. # of seeds selected

external memory depends on the diameter of the dataset. For example, for *com-lj*, the extra external memory needed is almost equal to the initial data size when $L = 2$, while for Dataset 4, the extra external memory needed increases much slower.

In addition to the parameter L, the size of extra external memory is also influenced by the number of seeds selected. Therefore, we set $L = 1$, and study the influence of the number of seeds on the size of extra external memory used. Figure 6 shows the extra memory ratio versus the percentage of number of seeds selected. We see that the external memory usage approximately increases linearly as the percentage of number of seeds increases.

Then, we study the influence of parameter L on the density of the densest subgraph. We set the number of seeds as 0.01% of the number of all vertices, and measure the density of the densest subgraph discovered with different values of parameter L. We define *density ratio of* L of a certain value of L as the ratio of the density of the densest subgraph found by the current value of L divided by the density of the densest subgraph discovered by setting L from 1 to the diameter of the initial graph. Figure 7 shows the density/densest density value versus the value of L. As the figure shows, for some of the datasets, it is intriguing to see that the quality of the subgraphs does not always increase as L increases. As L increases, the qualities slightly decrease in Dataset 2, Dataset 4 and Dataset 10, remain nearly constant in *CA-GrQc* and Dataset 3, but slightly increase in *com-lj*, *Email-Enron*, *Wiki-Vote*, *slash* and *CA-HepPh*. We can conclude that the qualities of the discovered subgraphs are relatively constant with increasing of L. Considering that L increase will not significantly improve performance, we just set $L = 1$ for measuring the extra external memory. As Figure 5 shows, the extra external memory is far less than the dataset itself when $L = 1$, and the data can be handled efficiently.

Then, we define the *density ratio of m* of a certain value of m as the ratio of the density of the densest subgraph found by the current value of L divided by the density of the densest subgraph found by setting m from 1 to the number of vertices in initial graph. Figure 8 shows the density ratio of m versus the number of seeds selected when $L = 1$. As the figure shows, for most of the datasets, we can get the best results using the top 0.01% suitable degree seeds. For Dataset 10 that performs slightly worse, it achieves relatively higher performance with top 0.01% suitable degree seeds, which are still far less than 0.01% of all the vertices. Therefore, we conclude that using less than 0.01% vertices as seeds is sufficient for all the

TABLE I
DESCRIPTION OF REAL-WORLD DATASETS

ID	Name	Description	$\|V\|$	$\|E\|$	Type
Dataset 1	Wiki-Vote [29]	Wikipedia who votes on whom network	7,115	207,378	small
Dataset 2	CA-GrQc [30]	Collaboration network of Arxiv General Relativity	12,008	237,010	small
Dataset 3	Email-Enron [31]	Enron company email list	36,692	367,662	small
Dataset 4	CA-HepPh [30]	Arxiv High Energy Physics paper citation network	34,546	421,578	small
Dataset 5	slash [14]	Slashdot social network from November 2008	77,360	905,468	small
Dataset 6	com-youtube [32]	Youtube online social network	1,134,890	2,987,624	large
Dataset 7	com-lj [32]	LiveJournal online social network	3,997,962	34,681,189	large
Dataset 8	com-orkut [32]	Orkut online social network	3,072,441	117,185,083	large

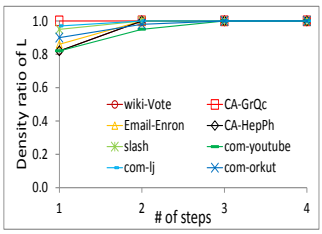


Fig. 7. The density ratio vs. # of transverse steps

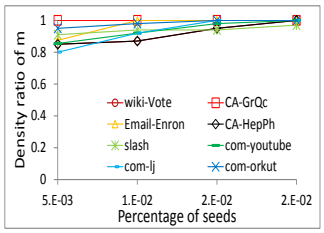
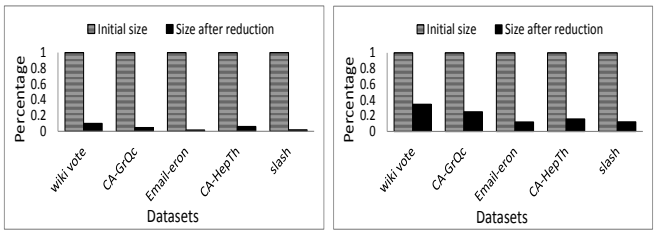


Fig. 8. The density ratio vs. # of seeds selected



(a) The # of vertices (b) The # of edges

Fig. 9. The size of the datasets before and after the reduction

datasets for the densest subgraph discovery.

Based on the experimental results of Figure 5, Figure 6, Figure 7 and Figure 8, we can conclude that for all the datasets, we only need to set $L = 1$ and use less than 0.01% of all the vertices to get almost the best results with the different parameter values. Since the number of seeds is small and $L = 1$, the extra external memory is less than 0.01% of the size of the initial dataset.

2) Algorithm efficiency and effectiveness on real-world dataSets: In this section, we evaluate the precision of the dentest subgraph discovery of our algorithm, and compare it with *GreedyAlg* and *PGreedyAlg* for discovering densest subgraph. First, we set the number of seeds as 0.01% of the number of all the vertices, $L = 1$, and run our algorithms on the datasets in Table I. Then, we compare the density of the densest subgraph discovered by our algorithm, *GreedyAlg* and *PGreedyAlg*. In *PGreedyAlg*, we set parameter $\varepsilon = 0$ for this algorithm to reach its best performance. Table II shows the comparison of the final results, where $\rho^2(G)$ presents the results of *PGreedyAlg*, $\rho^1(G)$ presents the results of *GreedyAlg*, and $\rho(G)$ presents the results of our algorithm. As shown in the table, our algorithm has the best performance on *CA-GrQc*, Dataset 2, Dataset 3, Dataset 4 and Dataset 10. *GreedyAlg* has the best performance in *CA-GrQc*, Dataset 2, and from *com-lj*, *Email-Enron*, *Wiki-Vote*, *slash* and *CA-HepPh*. We can see that our algorithm performs the best in 50% of the real datasets. Although our algorithm performs the worst in *Email-Enron*, *Wiki-Vote* and *CA-HepPh*, it still reaches at least 80% of the best results. *GreedyAlg* can achieve higher precision sometimes at the cost of low efficiency. However, *GreedyAlg* is not suitable for massive datasets due to the memory limitation, which is a critical requirement for Big Data. We emphasize the best performances with bold type in all the tables in the evaluation.

Another consideration for the comparison between our algorithm and other algorithms is the time efficiency. Since *GreedyAlg* cannot be parallelized and is not suitable for big datasets, we only compared the time efficiency between our algorithm and *PGreedyAlg* in the same situation. For the metrics, we not only compared the iteration times of the MapReduce process, but also compared the running time. Table IV shows the comparison of the number of MapReduce iterations and the execution time, where $i^2(G)$ presents the number of MapReduce iterations of *PGreedyAlg*, $i(G)$ presents the number of MapReduce iterations of our algorithm, $t^2(G)$ presents the execution time of *PGreedyAlg* and $t(G)$ presents the execution time of our algorithm. As shown in the table, our algorithm is terminated by 4 iterations since $L = 1$, while *PGreedyAlg* is terminated in at least 7 iterations. Also, our algorithm is much faster than *PGreedyAlg*. Therefore, we can conclude our algorithm is more time-efficient than *PGreedyAlg*. On average, we reduce the running time by 62%.

C. Performance of Exact Densest Subgraph Discovery Algorithm

In this section, we evaluate the performance of the M-O algorithm in comparison with ApproxMR by datasets in Table I. First, we evaluated the graph reduction phase, which focuses on two aspects: i) how many percent of vertices can be reduced from the initial graph size, and ii) how many MapReduce rounds are needed to reach a suitable size for the densest subgraph discovery phase. The first aspect evaluates the effectiveness of the graph reduction phase, which determines the feasibility of the densest subgraph discovery phase. If the dataset size after reduction is still big, the M-O algorithm cannot handle it in the second phase. The second aspect evaluates the efficiency of the M-O algorithm since one round of MapReduce process is time consuming. Second, we evaluated the density and the running time of the discovered densest subgraph. In Section V-C4, we measured the connectivity of the discovered densest subgraph. Finally, we used simulated datasets to evaluate the M-O algorithm in comparison with ApproxMR.

1) Comparison of sizes before and after reduction: Table III shows the comparisons of the number of vertices and the

TABLE II
THE DENSITY OF THE DENSEST SUBGRAPH FOUND BY DIFFERENT ALGORITHMS

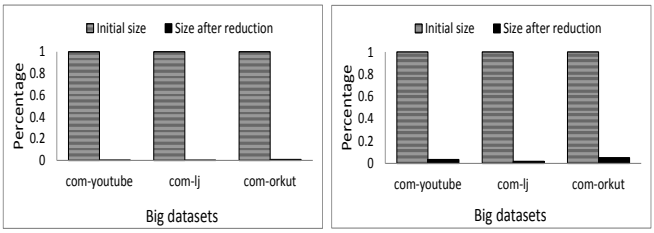
Datasets	ApproxMR	Approx	Heuristic	M-O	HHeuristic
Wiki-Vote	43.91	46.25	38.00	**49.2**	46.91
CA-GrQc	**22.39**	**22.39**	**22.39**	**22.39**	**22.39**
Email-Enron	35.31	**37.33**	32.09	**37.33**	37.12
CA-HepPh	30.05	**30.17**	25.42	**30.17**	**30.17**
slash	38.65	**42.27**	40.74	**42.27**	41.57
com-youtube	34.4	37.00	35.2	**38.6**	37.41
com-lj	35.31	37.33	36.26	**47.4**	42.38
com-orkut	176.2	182.5	184.00	**189.1**	185.73

TABLE III
COMPARISON OF DATASETS BEFORE AND AFTER REDUCTION

Datasets	# of vertices			# of edges		
	Before	After	After/Before	Before	After	After/Before
wiki-vote	7,115	727	10%	207,378	71,518	34%
CA-GrQc	12,008	123	1%	237,010	4,812	2%
Email-Eron	36,692	592	1%	367,662	44,182	12%
CA-HepTh	34,546	77	0.2%	421,578	1,964	0.4%
slash	77,360	1,417	1%	905,468	98,556	10%
com-youtube	1,134,890	1,685	0.1%	2,987,624	130,062	3%
com-lj	3,997,962	4,136	0.1%	34,681,189	650,724	1%
com-orkut	3,072,441	25,776	0.8%	117,185,083	9,800,872	8%

TABLE IV
THE COMPARISON OF THE NUMBER OF ITERATIONS AND EXECUTION TIME (UNIT/SECOND)

Datasets	Iterations				Time (second)			
	ApproxMR	Heuristic	M-O	HHeuristic	ApproxMR	Heuristic	M-O	HHeuristic
Wiki-Vote	9	**4**	7	5	367	**82**	187	102
CA-GrQc	8	**4**	7	5	482	**81**	172	117
Email-Enron	11	**4**	7	5	312	**84**	192	95
CA-HepPh	11	**4**	7	5	423	**88**	183	121
slash	11	**4**	7	5	514	**90**	207	153
com-youtube	14	**4**	7	5	740	**245**	310	289
com-lj	10	**4**	7	5	4014	**905**	2756	1485
com-orkut	12	**4**	7	5	26175	**9175**	11126	10754



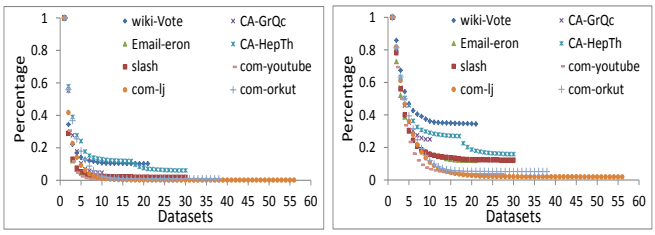
(a) The # of vertices (b) The # of edges

Fig. 10. The size of the datasets before and after the reduction for large datasets

number of edges of the datasets before and after the graph reduction phase by the M-O algorithm. In order to show the reduction performances clearly, we also show Figure 9 and Figure 10, which plot the ratio of the number of vertices and the number of edges before and after the reduction for the small datasets and large datasets, respectively. From Table III, Figure 9 and Figure 10, we see that the number of vertices after reduction is less than 1% of the initial number of vertices on average. Especially, for the large datasets (e.g., *com-youtube* and *com-lj*), the number of vertices after reduction is only about 0.1% of the initial number of vertices on average. The number of edges is also reduced to an average of 18% of the initial size for the small datasets. For the large datasets, the number of edges is only about 1% of the initial size. The reduction performance is even better for the large datasets. This large size reduction makes it possible to run the densest subgraph discovery phase in one computer, since the time complexity of this phase is determined by the number of vertices and edges.

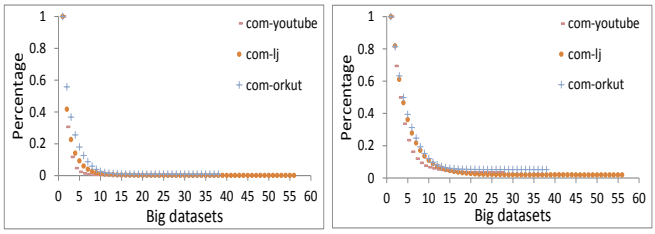
Take the large dataset *com-youtube* as an example, after the reduction, the number of vertices is 0.1% of the initial dataset and the number of edges is 3% of the initial dataset. If we use the push-relabel algorithm [33] for the densest subgraph discovery phase, which is the fastest algorithm for the min-cut max-flow problem with time complexity $O(|V||E|^2)$, then discovering the densest subgraph only takes 0.00009% ($|V||E|^2 = 0.1\% \times 3\% \times 3\%$) of the time for discovering the densest subgraph in the initial graph which can be easily calculated based on the time complexity. For the memory consumption, we only use about 3% of the memory for the initial graph since 97% ($1 - 3\% = 97\%$) of the edges have been deleted. Therefore, the large dataset reduction in the first graph reduction phase makes the dataset possible to be handled in the second densest subgraph discovery phase in the M-O algorithm.

2) The number of MapReduce rounds vs. data size: Figure 11 shows the percentage of the vertex size and the edge size of the remaining graph versus the number of MapReduce rounds in the graph reduction phase. It is interesting to see that the huge reductions only happened in the first a few rounds (5 rounds for most of the datasets). Also most of the datasets



(a) The size of reduced vertices (b) The size of reduced edges

Fig. 11. The size of reduced vertices and edges vs. the # of rounds



(a) The size of reduced vertices (b) The size of reduced edges

Fig. 12. The size of reduced vertices and edges vs. the # of rounds for large datasets

tend to be stable after 10 rounds of MapReduce processes. Figure 12 further shows the results for large datasets. After about 10 rounds of reductions, we can reduce the number of vertices to around 0.1% of the initial number of vertices and reduce the number of edges to around 1% of the initial number of edges. Although we can continue the graph reduction for more rounds for large datasets, the extra graph reduction does not help much in reducing the size of the dataset. Figure 12 indicates that the reduction performance is even better for the large datasets in which the data size can be reduced to less than 1% of the initial size in less than 10 rounds. This result indicates that the M-O algorithm is efficient in the graph reduction phase since the data can be reduced to the suitable size in only a few rounds.

3) Density of the discovered densest subgraph: In this section, we evaluate the performance of discovering the densest subgraph of the M-O algorithm in comparison with ApproxMR in terms of the density of the discovered densest subgraph. In ApproxMR, parameter ε is a slack variable for controlling the tradeoff between precision and efficiency. We set $\varepsilon = 0$ as in paper [9] to reach its best performance. Table II shows the density of the discovered densest subgraph in the M-O algorithm and ApproxMR. We bold the result for the better performance in comparison in all tables in the evaluation. As shown in the table, the M-O algorithm performs better almost for all the datasets while ApproxMR only performs as well as M-O algorithm on very small datasets (e.g., *CA-GrQc* and *CA-HepTh*). However, for small datasets, we do not need to apply the approximate algorithm since the exact algorithm can achieve both high efficiency and accuracy.

Table IV shows the comparison of the number of MapReduce iterations and the execution time, where $i^2(G)$ presents the number of MapReduce iterations of *PGreedyAlg*, $i(G)$ presents the number of MapReduce iterations of our algorithm, $t^2(G)$ presents the execution time of *PGreedyAlg* and $t(G)$ presents the execution time of our algorithm. As shown in the table, our algorithm is terminated by 4 iterations since $L = 1$, while *PGreedyAlg* is terminated in at least 7 iterations. Also, our algorithm is much faster than *PGreedyAlg*. Therefore, we can conclude our algorithm is more time-efficient than *PGreedyAlg*. On average, we reduce the running time by 62%.

4) Connectivity of the discovered densest subgraph: In this section, we measure the connectivity of the densest subgraph discovered by the M-O algorithm and ApproxMR. Figure 13 shows the percentage of connected subgraphs among the 8 discovered densest subgraphs from the 8 real-world datasets in the M-O algorithm compared to ApproxMR. Only 87.5% of the densest subgraph discovered by ApproxMR in all the datasets are connected. Although most of the densest subgraphs discovered by ApproxMR are connected, the lack of connectivity guarantee still influences its application. All the densest subgraphs discovered by the M-O algorithm are connected. This experimental result is consistent with our proved conclusion in Section IV-C. This result confirms that the M-O algorithm can guarantee the connectivity of the discovered densest subgraph.

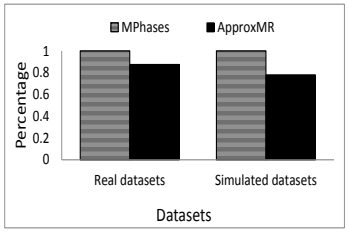
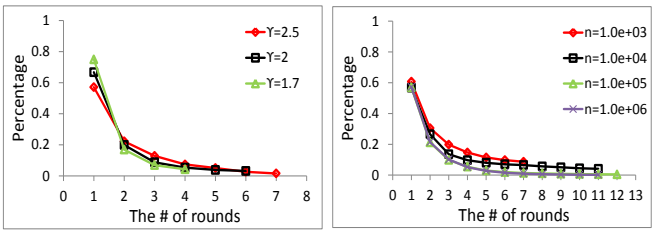


Fig. 13. Connectivity comparison

5) Efficiency of graph reduction on simulated natural graphs: Natural graphs usually follows a power law degree distribution with exponent parameter $\gamma \in (1, 3)$ [1]. Therefore, we show the percentage of the size of vertices in the remaining graph versus the number of rounds with different value of γ in $(1, 3)$ and different numbers of vertices separately in Figure 14.

Figure 14(a) shows the results of the simulated datasets with different degree power law parameter γ when we set the number of vertices in the datasets (denoted by n) to 1000. As γ increases, the required number of rounds for reducing the dataset to a suitable size for in-memory computing slightly increases. Figure 14(b) shows the results of the simulated datasets with different number of vertices n when we set $\gamma = 2.5$ which is a most common value for the normal natural graphs [1]. For all the simulated datasets with different sizes, the size of the dataset reduces quickly at the beginning and reaches less than 1% of the initial size only in 10 rounds. The sizes are reduced even faster for the datasets with bigger n. These phenomena are consistent with the phenomena in real-world datasets as in Figure 11 and Figure 12. Therefore, we conclude that the M-O algorithm is suitable for the big natural graphs with power law degree distribution and community features (which is different from the BA network introduced in section IV-C).

6) Density of the discovered densest subgraph on simulated natural graphs: Figure 15 shows the density of the discovered densest subgraph of the M-O algorithm and ApproxMR on each of the 50 randomly simulated graphs. As shown in the figure, we can see clearly that our algorithm can find denser subgraph comparing with ApproxMR. These results match the experimental results on real-world datasets in Table II.



(a) With different γ (b) With different n

Fig. 16. The correlation of the precision of final results and the percentage of neighbors which are small-degree nodes.

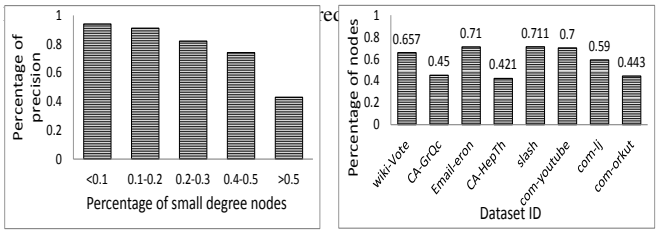


Fig. 17. The percentage of small-degree nodes deleted in first round.

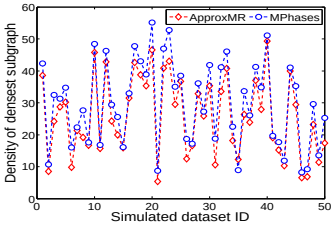


Fig. 15. The density of the discovered densest subgraph in 50 simulated datasets

7) Connectivity of the discovered densest subgraph on simulated natural graphs: We compare the connectivity of 50 densest subgraphs discovered by the M-O algorithm and ApproxMR in Figure 13. All of the densest subgraphs discovered by the M-O algorithm are connected, but there are 22% of the densest subgraphs discovered by ApproxMR which are disconnected.

D. A Hybrid Heuristic Algorithm

In this section, we compare the performances of heuristic algorithm and exact algorithm comprehensively. Then, we discuss the possibility of combining these two algorithms together to make a tradeoff between the performance of time efficiency and precision. Based on the discussion, we propose a Hybrid Heuristic (HHeuristic) algorithm. Finally, we evaluate the performance of HHeuristic and compare it with heuristic and M-O algorithms.

1) Heuristic algorithm vs. exact algorithm: As shown in Table II, although our proposed heuristic algorithm is as precise as the approximate algorithms like ApproxMR, the heuristic algorithm is still not competitive with the exact algorithm or even Approx. While at the same time, as shown as in Table IV, although the M-O algorithm is faster than ApproxMR, it is still not much slower and needs much more rounds of MapReduce process than the heuristic algorithm.

To sum up, the heuristic algorithm is fast but not precise enough, while M-O algorithm is precise but still need many rounds of MapReduce processes. In order to design an algorithm which can make a tradeoff between the time efficiency and the precision of the final results, we further

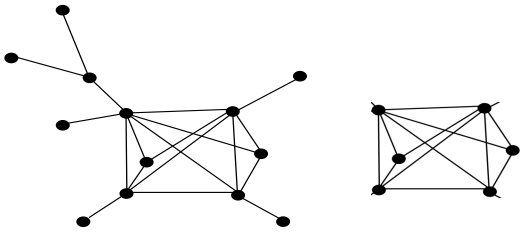


Fig. 18. An example of HHeuristic algorithm

analyze the reason that leads to the lower precision of heuristic algorithm. In Section IV-C, we proved that nodes with very few degrees are impossible to be in the densest subgraph. Intuitively, we assume that the lower precision of heuristic algorithm is caused by involving seeds with many neighbors with small degree. In order to verify our assumption, we define nodes with degree smaller than the average degree of the graph as small-degree nodes. Then, we measure the correlation of the precision of final results and the percentage of neighbors which are small-degree nodes as shown in Figure 16. As we can see from Figure 16, seeds with a larger percentage of neighbors which are small-degree nodes are related to low precisions. On the other hand, from the experiment of the M-O algorithm, we find that in the first round of the data reduction, more than half of the nodes with low degrees can be removed. In order to verify this assumption, we compare the percentage of small-degree nodes before and after the first round of data reduction as shown in Figure 17. As shown in Figure 17, the percentage of small-degree nodes can be decreased sharply in the first round of data reduction.

2) The Design of hybrid heuristic algorithm (HHeuristic): Based on the above analysis, we propose a hybrid heuristic algorithm for discovering densest subgraph with a tradeoff between time efficiency and precision. The basic idea is to delete all those nodes with few degrees by the data reduction process of the M-O algorithm. As shown in the left part of Figure 18, we can achieve it fast since more than half of the nodes can be deleted in the first round of the MapReduce process. Then, instead of keeping reducing the data in many rounds, we select eligible seeds and use the heuristic algorithm to find the dense subgraphs efficiently. As shown in the right part of Figure 18, we can get better results since the percentage of neighbors with one degree for each seed node has been significantly decreased.

3) The performance of hybrid heuristic algorithm: Table II shows the density of the discovered densest subgraph in the heuristic, HHeuristic and M-O algorithms. As shown in the table, the densities of densest subgraph discovered by HHeuristic is close to the densities of the densest subgraphs discovered by the M-O algorithm, which is much higher than the densities of densest subgraph discovered by the heuristic algorithm. Table IV shows the density of the discovered densest subgraph in the heuristic, HHeuristic and M-O algorithms. As shown in the table, the densities of the densest subgraph discovered by HHeuristic is close to the densities of the densest subgraphs discovered by the M-O algorithm, which is much higher than the densities of the densest subgraph discovered by heuristic algorithm.

VI. Conclusion

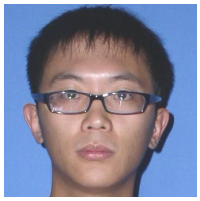
In this paper, we studied the densest subgraph problem by designing two different algorithms based on different features that natural graphs have. First, by analyzing the features of natural graphs, we designed a heuristic algorithm for discovering the connected densest subgraph for massive undirected graphs in a MapReduce framework by taking advantage of the features of natural graphs. Second, we proposed an exact algorithm for big data for the problem of discovering the densest subgraph. Experimental results show that our algorithms are more time-efficient and precise than other algorithms. In the future, we will explore taking advantages of the features of natural graphs to efficiently discover multiple dense subgraphs, overlapping dense subgraphs, dense subgraphs in different scales according to the necessities of different applications.

VII. Acknowledgements

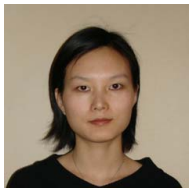
This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751. Early versions of this work were presented in the Proceedings of ICCCN [34] and the Proceedings of NAS [35].

References

[1] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical Review*, vol. E 69, 2004.
[2] L. Wan, B. Wu, N. Du, Q. Ye, and P. Chen, "A new algorithm for enumerating all maximal cliques in complex network.," *ADMA*, vol. 4093, 2006.
[3] A. V. Goldberg, "Finding a maximum subgraph," *Technical report*, 1984.
[4] L. Yan, H. Shen, and K. Chen, "TSearch: Target-oriented low-delay node searching in dtns with social network properties.," in *Proc. of INFOCOM*, 2015.
[5] M. Han, M. Yan, J. Li, S. Ji, and Y. Li, "Neighborhood-based uncertainty generation in social networks.," *Journal of Combinatorial Optimization.*, vol. 28, pp. 561–576.
[6] M. Han, M. Yan, Z. Cai, Y. Li, X. Cai, and J. Yu, "Influence maximization by probing partial communities in dynamic online social networks.," *Transactions on Emerging Telecommunications Technologies*, pp. 561–576.
[7] B. Saha, A. Hoch, S. Khuller, L. Raschid, and X.-N. Zhang, "Dense subgraphs with restrictions and applications to gene annotation graphs," in *RECOMB*, vol. 6044, pp. 456–472, Springer, 2010.
[8] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph.," in *APPROX*, vol. 1913 of *Lecture Notes in Computer Science*, pp. 84–95, Springer, 2000.
[9] B. Bahmani, R. Kumar, and S. Vassilvitskii, "Densest subgraph in streaming and mapreduce," *CoRR*, vol. abs/1201.6567, 2012.
[10] M. Han, J. Li, and Z. Zou, "K-close: Algorithm for finding the close regions in wireless sensor networks based uncertain graph mining technology.," *Journal of Software*, vol. 22, pp. 131–141, 2011.
[11] S. Milgram, "The small world problem," *Psychology Today*, vol. 61, pp. 60–67, 1967.
[12] A. L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999.
[13] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, no. 393, pp. 440–442, 1998.
[14] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters.," *Internet Mathematics*, vol. 6, pp. 29–123, 2009.
[15] S. Zhou and R. J. Mondragn, "The rich-club phenomenon in the internet topology.," *IEEE Communications Letters*, vol. 8, pp. 180–182, 2004.
[16] C. Seshadhri, T. G. Kolda, and A. Pinar, "Community structure and scale-free collections of er graphs," *CoRR*, vol. abs/1112.3644, 2011.
[17] S. Khuller and B. Saha, "On finding dense subgraphs," *ICALP*, vol. 14, 2009.
[18] U. Feige, G. Kortsarz, and D. Peleg, "The dense k-subgraph problem.," *Algorithmica*, vol. 29, pp. 410–421, 2001.
[19] Y. Asahiro, R. Hassin, and K. Iwama, "Complexity of finding dense subgraphs," *Discrete Applied Mathematics*, vol. 121, pp. 15–26, 2002.
[20] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *in Proc. of VLDB*, pp. 721–732, 2005.
[21] J. Chen and Y. Saad, "Dense subgraph extraction with application to community detection," *IEEE Trans. Knowl. Data Eng.*, vol. 24, pp. 1216–1230, 2012.
[22] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," in Proc. of WWW, pp. 1481–1493, 1999.
[23] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun ACM*, vol. 51, pp. 107–113, 2008.
[24] V. Adamchik and H. M. Srivastava, "Some series of the zeta and related functions," *Analysis*, vol. 18, pp. 131–144, 1998.
[25] "Apache hadoop," 2015. http://hadoop.apache.org/.
[26] "Stanford network analysis project," 2013. https://snap.stanford.edu/.
[27] M. Newman, "The structure and function of complex networks," *Review, SIAM*, 2003.
[28] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos, "Doulion: counting triangles in massive graphs with a coin," in *Proc. of KDD*, pp. 837–846, ACM, 2009.
[29] J. Leskovec, D. P. Huttenlocher, and J. M. Kleinberg, "Signed networks in social media," *CoRR*, vol. abs/1003.2424, 2010.
[30] J. Leskovec, J. Kleinberg, and C. Faloutsos, "Laws of graph evolution: densification and shrinking diameters," *Knowledge Discovery*, vol. 1, pp. 1–40, 2006.
[31] B. Klimt and Y. Yang, "Introducing the enron corpus," in *CEAS*, 2004.
[32] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *CoRR*, vol. abs/1205.6233, 2012.
[33] B. V. Cherkassky and A. V. Goldberg, "On implementing the push-relabel method for the maximum flow problem.," *Algorithmica*, vol. 19, no. 4, pp. 390–410, 1997.
[34] B. Wu and H. Shen, "Discovering the densest subgraph in mapreduce for assortative big natural graphs.," in *Proc. of ICCCN Workshop on BDeHS*, pp. 35–41, 2015.
[35] B. Wu and H. Shen, "A time-efficient connected densest subgraph discovery algorithm for big data.," in *Proc. of NAS*, pp. 55–59, 2015.



Bo Wu received both his BS and MS degree in Computer science from Northwestern Polytechnic University, China in 2009 and 2011 respectively. He is currently a Ph.D student in the Department of Electrical and Computer Engineering of Clemson University. His research interests include social networks and big graph data analysis.



Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an associate professor in the Department of Computer Science, University of Virginia. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.